

**МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»**

**(РУТ (МИИТ)**

|  |
| --- |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Кафедра «Управление и защита информации»

**Отчет по заданию**«Алгоритм Штрассена»

**по дисциплине**

«Методы программирования»

**Выполнил:** студент

группы ТКИ-311

Станчук П.Н.

**Проверил:** доценты кафедры УиЗИ, к.т.н. Логинова Л. Н. и

Сафронов А.И.

**Москва 2022 г.**

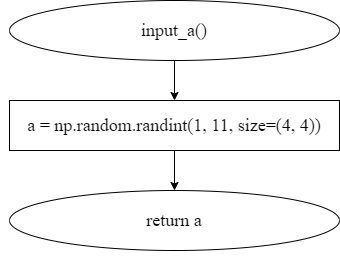
**1. Цель работы**

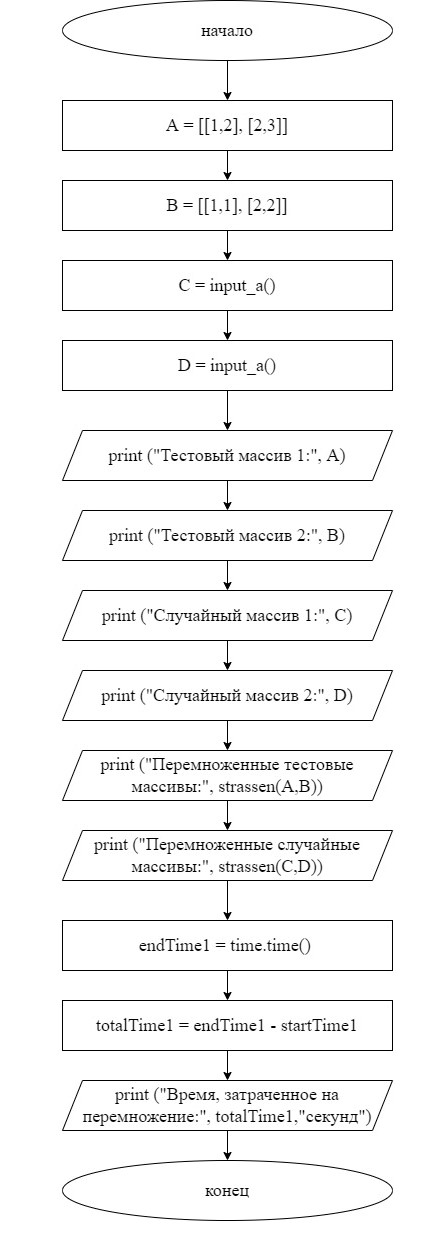
Изучение алгоритма Штрассена.

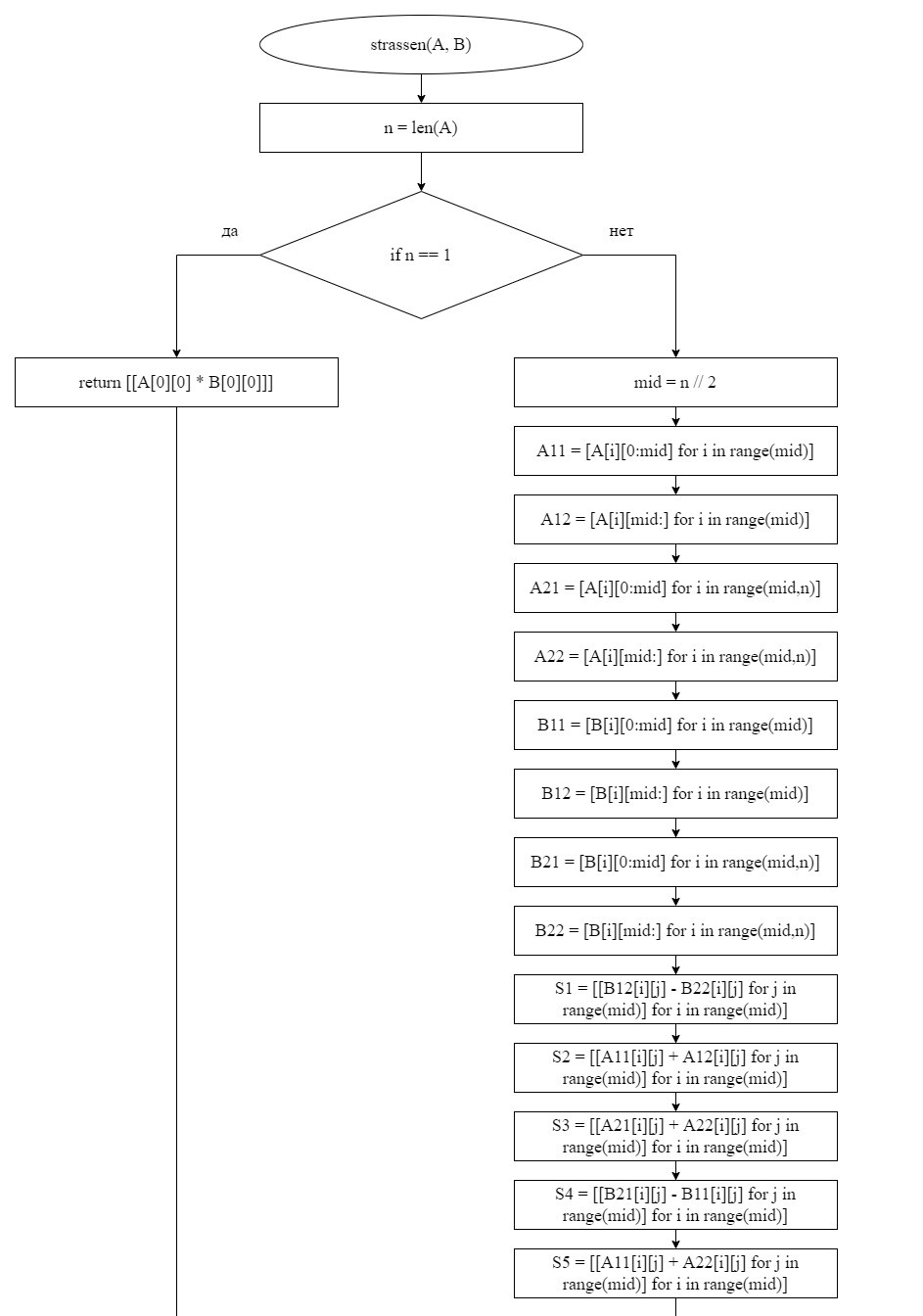
**2. Формулировка задачи**

Реализовать программу, выполняющую алгоритм Штрассена.

**3. Блок-схема алгоритма**

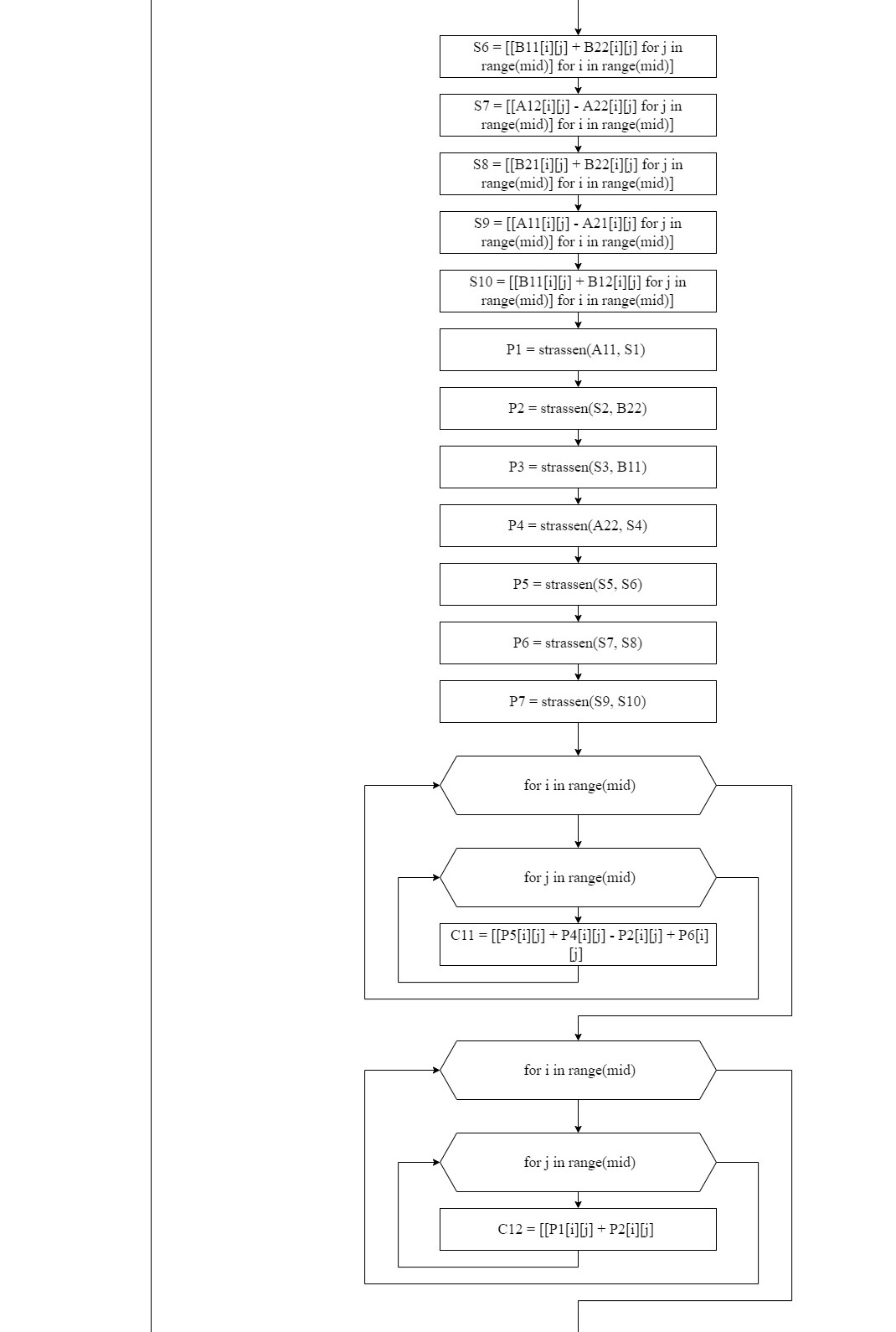




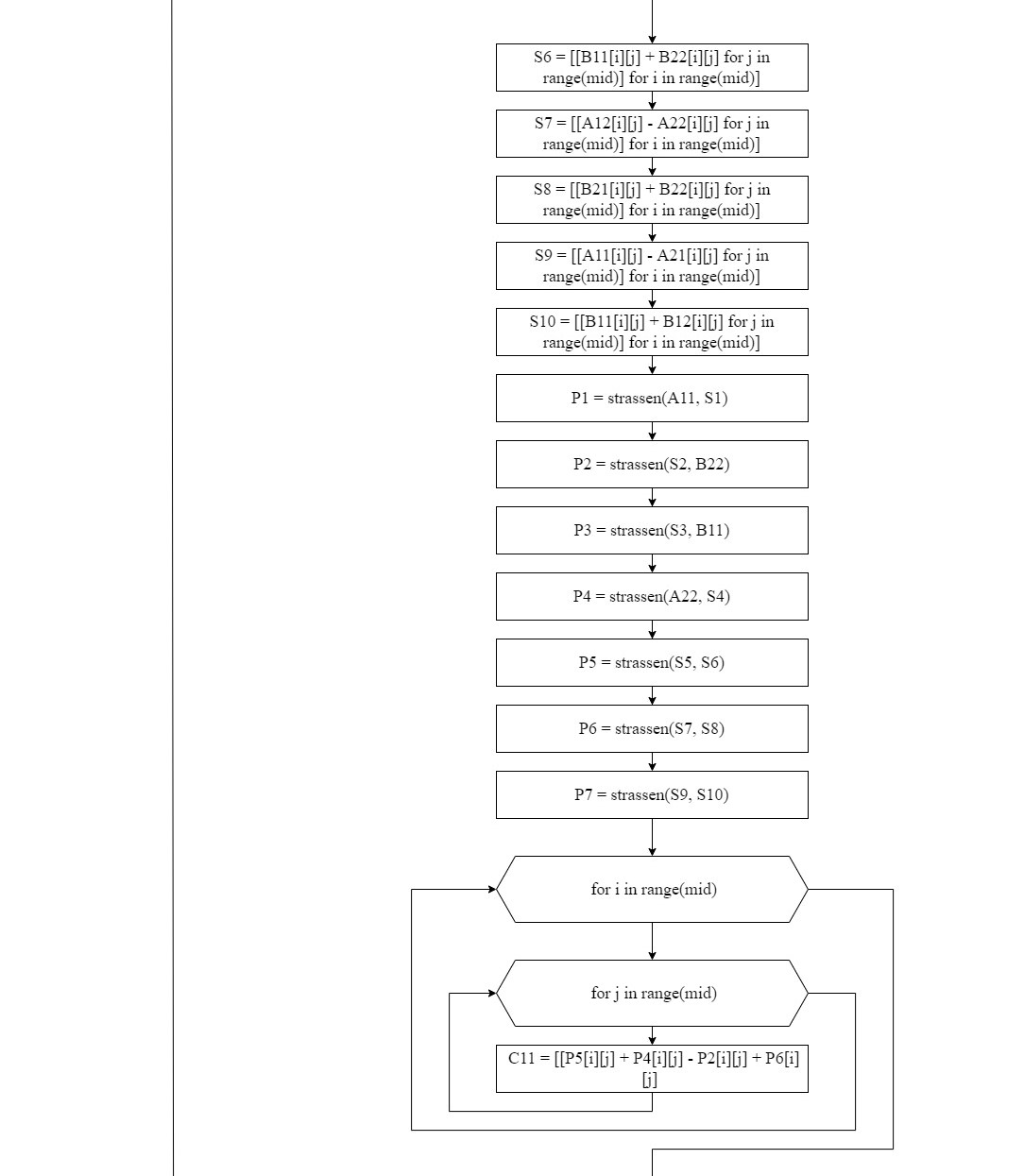


**1 2**

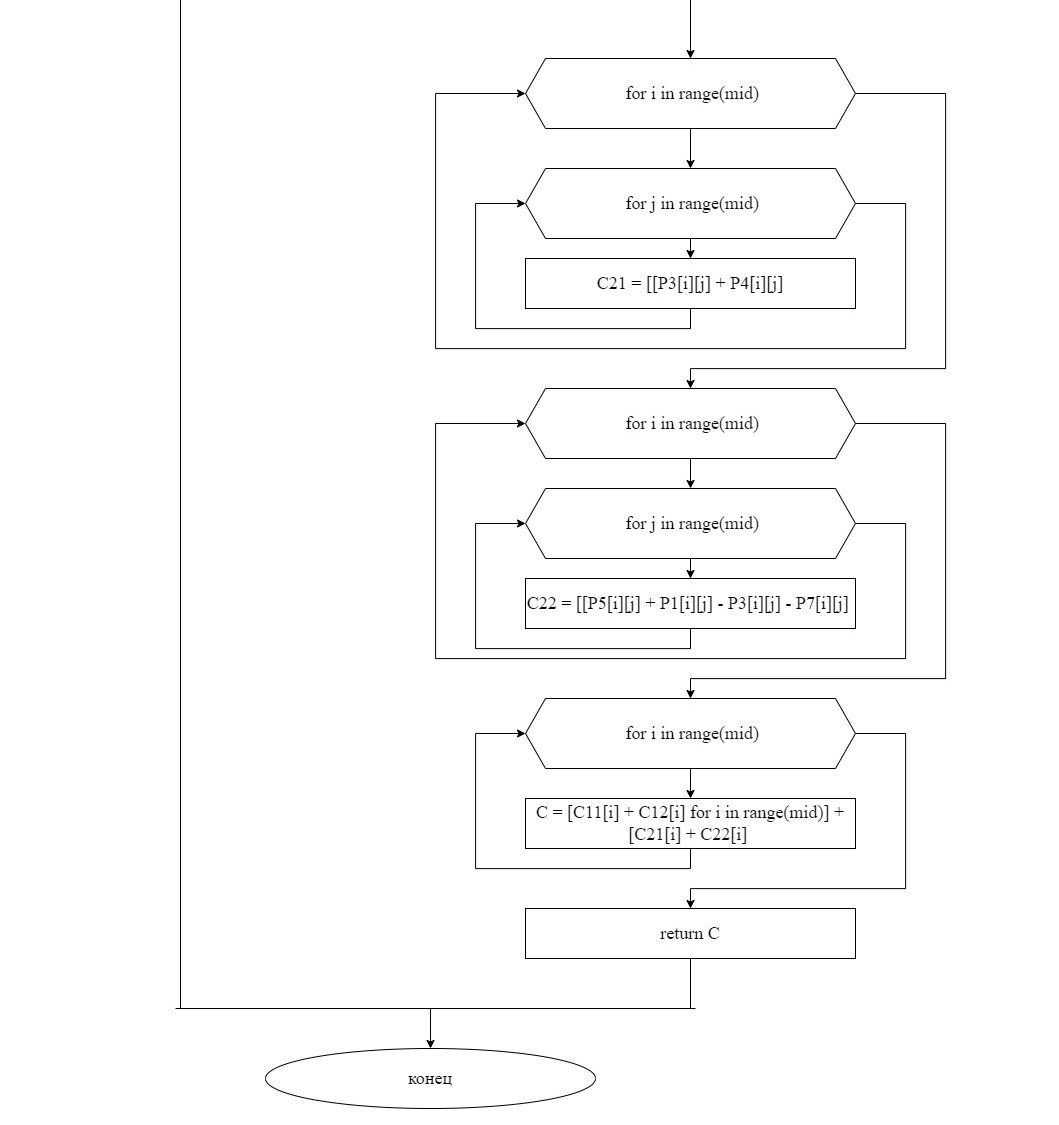
**1 2**



**3 4**

 **3 4**

**5 6**

 **5 6**

**4. Подбор тестовых примеров**

Ввод: 2 матрицы размера 2х2

A = [[1,2], [2,3]]

B = [[1,1], [2,2]]

Вывод: произведение массивов С = [[5,5], [8,8]].

**5. Код программы:**

import time

import numpy as np

startTime1 = time.time()

def strassen(A, B):

n = len(A)

if n == 1:

return [[A[0][0] \* B[0][0]]]

else:

mid = n // 2

A11 = [A[i][0:mid] for i in range(mid)]

A12 = [A[i][mid:] for i in range(mid)]

A21 = [A[i][0:mid] for i in range(mid,n)]

A22 = [A[i][mid:] for i in range(mid,n)]

B11 = [B[i][0:mid] for i in range(mid)]

B12 = [B[i][mid:] for i in range(mid)]

B21 = [B[i][0:mid] for i in range(mid,n)]

B22 = [B[i][mid:] for i in range(mid,n)]

S1 = [[B12[i][j] - B22[i][j] for j in range(mid)] for i in range(mid)]

S2 = [[A11[i][j] + A12[i][j] for j in range(mid)] for i in range(mid)]

S3 = [[A21[i][j] + A22[i][j] for j in range(mid)] for i in range(mid)]

S4 = [[B21[i][j] - B11[i][j] for j in range(mid)] for i in range(mid)]

S5 = [[A11[i][j] + A22[i][j] for j in range(mid)] for i in range(mid)]

S6 = [[B11[i][j] + B22[i][j] for j in range(mid)] for i in range(mid)]

S7 = [[A12[i][j] - A22[i][j] for j in range(mid)] for i in range(mid)]

S8 = [[B21[i][j] + B22[i][j] for j in range(mid)] for i in range(mid)]

S9 = [[A11[i][j] - A21[i][j] for j in range(mid)] for i in range(mid)]

S10 = [[B11[i][j] + B12[i][j] for j in range(mid)] for i in range(mid)]

P1 = strassen(A11, S1) # Compute P1-P7

P2 = strassen(S2, B22)

P3 = strassen(S3, B11)

P4 = strassen(A22, S4)

P5 = strassen(S5, S6)

P6 = strassen(S7, S8)

P7 = strassen(S9, S10)

C11 = [[P5[i][j] + P4[i][j] - P2[i][j] + P6[i][j] for j in range(mid)] for i in range(mid)]

C12 = [[P1[i][j] + P2[i][j] for j in range(mid)] for i in range(mid)]

C21 = [[P3[i][j] + P4[i][j] for j in range(mid)] for i in range(mid)]

C22 = [[P5[i][j] + P1[i][j] - P3[i][j] - P7[i][j] for j in range(mid)] for i in range(mid)]

C = [C11[i] + C12[i] for i in range(mid)] + [C21[i] + C22[i] for i in range(mid)]

return C

def input\_a():

a = np.random.randint(1, 11, size=(4, 4))

return a

A = [[1,2], [2,3]]

B = [[1,1], [2,2]]

C = input\_a()

D = input\_a()

print ("Тестовый массив 1:", A)

print ("Тестовый массив 2:", B)

print ("Случайный массив 1:", C)

print ("Случайный массив 2:", D)

print ("Перемноженные тестовые массивы:", strassen(A, B))

print ("Перемноженные случайные массивы:", strassen(C, D))

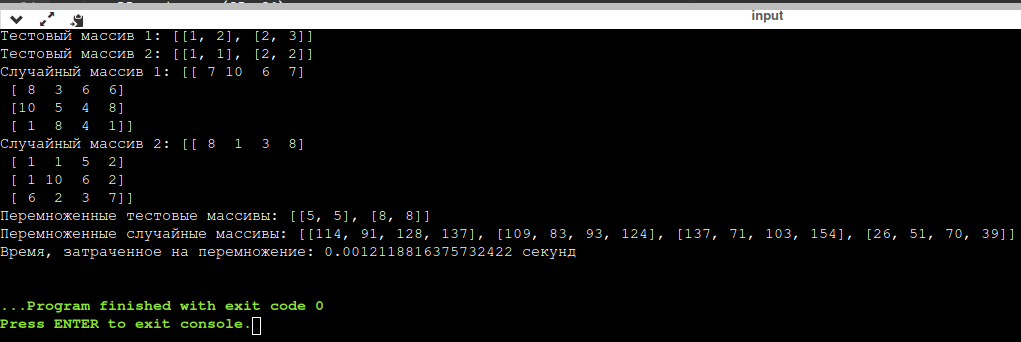
endTime1 = time.time() #время конца замера

totalTime1 = endTime1 - startTime1 #вычисляем затраченное время

print ("Время, затраченное на перемножение:", totalTime1, "секунд")

**6. Результат расчёта тестовых примеров на ПК**

Проверим работоспособность алгоритма



Тестовые массивы перемножены верно.

Сравним время работы алгоритма. Для этого закомментируем обработку тестовых массивов.

На перемножение массивов размерности 4х4 каждый затрачено 0.0010876655578613281 секунд.

На перемножение массивов размерности 8х8 каждый затрачено 0.0038132667541503906 секунд, то есть при увеличении количества элементов матриц в 4 раз время увеличилось приблизительно в 3.5 раза.

**7. Вывод**

В данной работе я реализовала алгоритм Штрассена на Python и протестировала его, а также узнала, что при увеличении количества количества элементов матриц в 4 раз время увеличилось приблизительно в 3.5 раза.